2009 15th International Conference on Parallel and Distributed Systems

Using Grid Middleware to Query a Heterogeneous Distributed Version of the SDSS Database

Helen X Xiang
Computer Science, University of Hertfordshire, UK
h.xiang@herts.ac.uk

Abstract

This paper explores the use of Grid technologies for the manipulation of the Sloan Digital Sky Survey (SDSS) database. The paper first provides background information on the SDSS database, OGSA-DAI operations and OGSA-DQP operators. We then analyse the execution of OGSA-DQP for querying a heterogeneous distributed version of the SDSS database. In particular, we examine the different SOAP interactions between OGSA-DQP components and look at simple queries plans. After modifying the OGSA-DQP coordinator we successfully ran OGSA-DQP queries against the distributed SDSS database.

1. Introduction

The *Sloan Digital Sky Survey* (SDSS) is a project that has built a very detailed digital map of the visible stars and galaxies in the night sky [14]. The data produced by the survey is summarised in a multi-terabyte relational database containing photometric objects and spectroscopic information. The SDSS database is available to the scientists and the public via the SkyServer (http://skyserver.sdss.org) or various mirror sites (including one we set up in the University of Portsmouth).

In recent papers [8, 9, 10, 13] we described how we created an experimental distributed version of the SDSS DR5 database, using Grid middleware. This is based on OGSA-DQP (Open Grid Services Architecture—Distributed Query Processing), developed by the University of Manchester and the University of Newcastle upon Tyne [7]. OGSA-DQP is in turn based on the OGSA-DAI middleware, developed by the Edinburgh Parallel Computing Centre (EPCC) and the UK National e-Science Centre (NeSC) [6]. We used OGSA-DAI and OGSA-DQP to integrate the data across different sites—forming a logical distributed database system. Global distributed queries can be processed over this logical database system [11].

This paper pays partial attention to the insight of the execution of OGSA-DQP query. It examines the OGSA-DAI operations and OGSA-DQP operators and describes the different SOAP interactions between OGSA-DQP components. We looked into the query plan for a simple query and made modifications and improvements in order to run OGSA-DQP queries against the SDSS database. Please refer to [13] for the details of the OGSA-DQP workflow and the interactions among its components. This paper follows on from our earlier papers [8, 9, 10, 13].

2 The OGSA-DAI Operations and OGSA-DQP Operators

The OGSA-DAI project [6] is one of the UK e-Science projects that develop Grid middleware to support data access and integration across separate resources in the Grid environment. OGSA-DAI supports a number of DBMSs including Microsoft SQL Server and Oracle, and is therefore an appropriate vehicle to integrate datasets that are distributed over multiple sites.

Table 1 shows some sample OGSA-DAI operations. Apart from supporting the access to the underlying data sources through the exposed Data Service Resource, a Data Service supplies a number of functions for providing service information.

A typical OGSA-DAI session might involve several steps. The first might be to list all the available Data Service Resources and acquire their IDs using the `ListResources` operation (an empty list means there is no data source exposed by the Data Service). With the required Data Service Resource located, the perform document is made up with a number of activities, then submitted to the target Data Service Resource for processing using the `Perform` operation. These activity requests are then processed by the target Data Service Resource to apply the data-related actions on the associated data source. The Data Service Resource retrieves the data result set (if any) from the underlying data source. Results may be obtained

1521-9097/09 $26.00 © 2009 IEEE
DOI 10.1109/ICPADS.2009.141

Operation Types	Operation Names	Input	Outputs
Data	Perform	Perform document	Response document
	GetFully	- session name - output stream	dataset
	GetBlocks	- session name - output stream (number of data blocks=1)	A data block (a large dataset)
	GetNBlocks	- session name - output stream (number of data blocks=N)	N data blocks (N large datasets)
	PutFully	- session name - input stream - dataset	(None)
	PutBlock	- session name - input stream - data block (large dataset)	(None)
Service	ListResources	(None)	List of Data Service Resource(s) ID(s)
	GetVersion	(None)	OGSA-DAI version
Property and State	GetResourceProperty	- Data Service Resource handle	Property value of the Data Service Resource
	GetMultipleResourceProperties	- Data Service Resource(s) handle(s)	Property value of the Data Service Resource(s)
	QueryResouceProperties (OGSA-DAI WSRF only)	- XPath query	Property value of Data Service Resource(s) (XML)

Table 1. Some OGSA-DAI operations

with operations like `GetNBlocks`. As well as querying the data source, OGSA-DAI also supports database updating, and bulk loading data from one table to another. The configuration file for a Data Service Resource specifies the set of activities that the resource supports.

The OGSA-DQP system is a service based distributed query processor for planning, scheduling and executing distributed queries in parallel [4, 1, 2, 5]. OGSA-DQP evaluates against distributed data sources that are exposed by OGSA-DAI data service resources. We used OGSA-DQP 3.2 *Tech Preview* in this research. A final version of OGSA-DQP 3.2, was released during the latter stages of the research and we expect most of our conclusions would carry over to the new release.

OGSA-DQP supports MySQL and simple databases [3]. We extended its data type support to handle a complex scientific database schema in SQL Server and Oracle [13].

OGSA-DQP introduces two different types of services: coordinator and evaluator. The *DQP coordinator* interacts with the client applications, and also parses and schedules distributed query executions. The DQP evaluator executes the actual query. (See [13] for more details.)

The different kinds of operator supported by the DQP evaluator are listed in Table 2. Each partition in a query plan is a tree of these operators. Each implementation class consumes zero or more input streams of tuples, and outputs a stream of tuples. Based on the source code of the OGSA-DQP 3.2 Tech Preview, we outline the behaviour of the implementation classes.

`TableScanOp` executes a SQL query, reading rows of a single table that match the specified predicate (if any). *All* columns of every selected row are returned. Streams out result row-set.

`HashJoinOp` is an implementation of join that works by reading the full left input row-set into a hash table in the evaluator, then streaming in the right input row-set, joining individual rows by lookup in the hash table as they arrive, and streaming out the result rows (if any).

`UnionOp` is an implementation of relation union. It streams in the left input row-set, row by row, and streams them out. It then does the same to the right input row-set.

`ReduceOp` is an implementation of project and aggregate operations on a relation. It streams in the input row-set, row by row. In the "project" case, it streams out the specified columns of the row. In the "aggregate" case, it internally accumulates a sum, minimum, maximum, count,

Operator	Implementation class
TABLE_SCAN	TableScanOp
HASH_JOIN	HashJoinOp
HASH_LOOPS	HashLoopJoinOp
THETA_JOIN	ThetaJoinOp
CARTESIAN_PRODUCT	CartesianProductOp
ORDER_BY	OrderByOp
UNION	UnionOp
APPLY	ReduceOp
OPERATION_CALL	OperationCallOp *or* UserDefinedFunctionOp
PRINT	PrintOp
EXCHANGE	ExchangeOp
UNNEST	UnnestOp

Table 2. OGSA-DQP evaluator operators

standard deviation, or average, and outputs the results when the input stream ends.

PrintOp prints out the input row-set. It streams it in row by row, and accumulates it into buffer-sized fragments of XML. These are fed to the coordinator QueryExecutionProcessor by calls to its putData() method. This putData() method will convert the XML fragment to a WebRowSet, which is passed to the output of the DQPQueryStatementActivity.

ExchangeOp is a communication operation. In an evaluator that is a *producer* in the exchange, the input row set is streamed in and sent in blocks to one or more evaluators that are *consumers* in the exchange. In an evaluator that is a consumer in the exchange, the result of the exchange operator is a union of the row-sets sent by one or more producers in the exchange. In a producer, the destination for an individual block is determined by the class called Arbitrator, and controlled by the arbitrator policy associated with the exchange operation. Data is sent to another evaluator service through the sendData SOAP operation exposed by the service. Data is sent to the root evaluator through the putBlock SOAP operation method on the coordinator data service resource. In either case (and in the case where the destination is the same evaluator as the source) the message is tagged with the destination ExchangeOp. The incoming data is queued before being streamed out from that ExchangeOp.

The other operations: HashLoopJoinOp, ThetaJoinOp, CartesianProductOp, OrderByOp, OperationCallOp and UserDefinedFunctionOp are less relevant to our current work—please refer to [11] for the behaviours of their implementation classes.

In the following sections, we run a few queries using some of the OGSA-DQP operators.

3 A Reduced SDSS Distributed over a Local Area Network

To set the scene for running the OGSA-DQP queries, this section briefly describes how we distributed a reduced SDSS database called MyBestDR5 among three hosts in different university buildings. Two of the hosts are running Oracle $10g$, while the other host is running Microsoft SQL Server.

In the original MyBestDR5 database, objID ranged from $587,726,014,001,184,891$ to $588,848,901,530,387,496$. We used objID to split its PhotoObjAll schema into three parts:

- host 1:

$$\text{ObjId} \leq 587,726,015,614,000,000$$

- host 2:

$$\text{ObjId} > 587,726,032,254,888,888$$

- host 3:

$$587,726,015,614,000,000 < \text{ObjId} \leq 587,726,032,254,888,888$$

Tables 3 illustrates the numbers of records of tables in the MyBestDR5 PhotoObjAll schema using the objID partition.

We deployed the migrated SDSS Oracle schema on the two Oracle databases (on hosts 1 and 2). Using the Windows BCP utility, we extracted two set of relevant data files from the orignal MyBestDR5 database in SQL Server, with objID partitioning conditions. Using the SQL*Loader, we injected the appropriate data files to databases on hosts 1 and 2.

We then removed these records from the copy of MyBestDR5 database on the orignal SQL Server database using the objID partitioning conditions—the cut down version of MyBestDR5 database formed the third partition (on host 3).

We now have a distributed SDSS MyBestDR5 database and we want to expose it using the OGSA-DAI middleware and configure it with the OGSA-DQP toolkit before running the distributed queries.

We installed an OGSA-DAI WSRF 2.2 instance on three hosts and deployed an OGSA-DAI data service on each host in an Apache Tomcat Web server using port 8080.

We then configured the OGSA-DAI data service resources to expose the SDSS databases on the three hosts via those OGSA-DAI data services.

At this point the SDSS MyBestDR5 database is distributed among three hosts—two Oracle and one SQL

Table name	Number of rows			
	MyBest-DR5	host 1	host 2	host 3
PhotoObjAll	200,276	68,118	66,856	65,302
BestTarget2-Sector	9,255	3,108	3,065	3,082
First	163	48	50	65
Masked-Object	2,095	587	1,138	370
Match(obj-ID1)	22,276	5,482	8,807	7,987
MatchHead	11,126	5,457	10	5,659
Neighbors	1,057,304	355,214	356,819	345,271
ObjMask	193,996	65,972	64,749	63,275
PhotoAux-All	200,276	68,118	66,856	65,302
Photo-Profile	6,981,758	2,362,382	2,335,263	2,284,113
PhotoTag	200,276	68,118	66,856	65,302
Photoz	200,276	68,118	66,856	65,302
Photoz2	58,225	19,709	19,622	18,894
ProperMo-tions	25,072	8,259	8,560	8,253
Rosat	323	81	104	138
USNO	21,422	7,472	7,243	6,707
Zone	162,657	55,183	54,316	53,158

Table 3. Number of photometric object records in the `MyBestDR5 PhotoObjAll` schema (partitioned by objID column)

Server—with the distributed SDSS data resources exposed via the OGSA-DAI middleware.

To query the distributed SDSS `MyBestDR5` database, we installed the OGSA-DQP 3.2 (Tech Preview) toolkit on the distributed sites. On each host, we deployed an OGSA-DQP evaluator service on a Tomcat instance that is separate from the Tomcat used by the OGSA-DAI data service deployed earlier on. An OGSA-DQP coordinator can be installed on one of the three distributed SDSS hosts on the OGSA-DAI Data Services deployed earlier on. It can also be installed on a separate host. The client application can then interact with the OGSA-DQP coordinator service to create an OGSA-DQP coordinator instance for executing queries.

Figure 1 summarizes the architecture of the distributed SDSS `MyBestDR5` database system developed in this section. Our experience running queries against this distributed database will be described in the following sections.

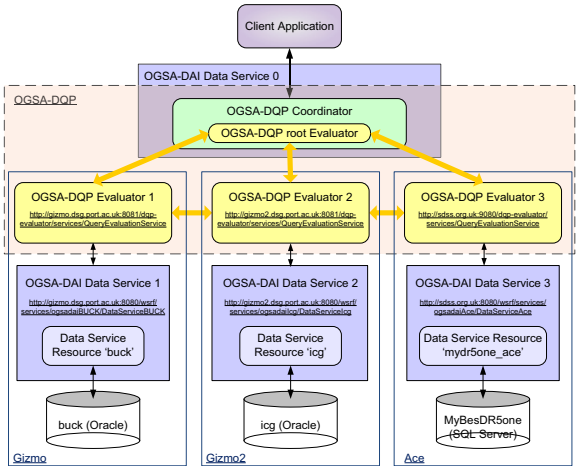

Figure 1. Distributing the SDSS `MyBestDR5` database among three hosts

4. Execution of an OGSA-DQP Query

Having successfully generated the OGSA-DQP SDSS coordinator instance in [13], we tried to run some OGSA-DQP queries against the SDSS `MyBestDR5` database.

We first look at this simple SQL Query on `MyBestDR5` database:

```
SELECT match, id, ObjID
FROM First
WHERE objID=587726016148734065
```
(1)

This query returns 1 row in less than one second.

A comparable OGSA-DQP query against the SQL Server partition of our distributed database is:

```
SELECT match, id, ObjID
FROM MyBestDR5one_First
WHERE objID=587726016148734065
```
(2)

For the Oracle database on the first host a similar query is:

```
SELECT MATCH, ID, OBJID
FROM buck_FIRST
WHERE OBJID=587726014001315907
```
(3)

In OGSA-DQP syntax, prefixes like `MyBestDR5one_` and `buck_` identify the partitions against which the queries will execute. The `ObjID` values select tuples in those partitions.

The `WHERE` clauses select different rows. This reflects the different subsets of rows held on the two hosts. We are essentially running the above two *local* queries through OGSA-DQP, against *one* underlying database (SQL Server database *or* Oracle database).

The OGSA-DQP query against the SQL Server ran successfully but query 3 against the Oracle database did not.

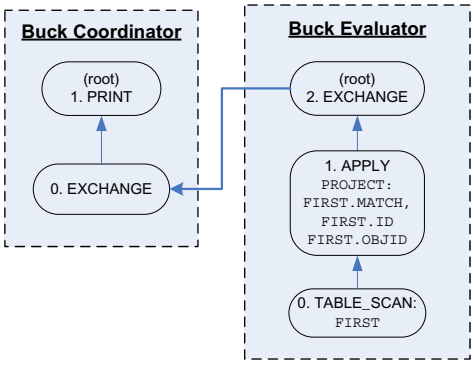

Figure 2. Query plan for query 3 running on host 1

The eventual resolution of this problem involves modifying OGSA-DQP coordinator and OGSA-DAI classes to deal more appropriately with Oracle data type `number`. For further technical details see [13]. Of more interest for purpose of current paper are lessons that we learned while tracing these problems since they providing insight into the execution of OGSA-DQP queries.

From examination of the OGSA-DQP source code we discovered that the types embedded in the WebRowSet were controlled by inputs to a `PRINT` operator in the evaluator. But we needed additional information about the query execution to determine those inputs.

To get more information about states of execution during the query, we captured all the SOAP messages exchanged using the Axis monitoring program, `tcpmon`. This information gives an insight into workings of OGSA-DQP, so we illustrate the captured pattern of messages in the following figures: 3, 4. This pattern can be understood by reference to the query plan, which we later captured from the OGSA-DQP coordinator log, by enabling DEBUG level logging. The query plan is an XML document. Figure 2 is a graphical representation of query plan structure.

The query plan shown in Figure 2 contains two partitions. The partition labelled *Buck Coordinator* is executed by the *root evaluator* which is actually part of the DQP coordinator (see Figure 1). The partition labelled *Buck Evaluator* is executed in the real DQP evaluator service. Each numbered node in a partition represents an *operator* in the query plan. The `TABLE_SCAN` operator reads selected rows from the database via the data service resource. The `APPLY` operator projects the requested columns from those rows—this operation is internal to the DQP evaluator. The `EXCHANGE` operator communicates between evaluators in general. In this case, it just sends the data to the root evaluator inside the coordinator. Finally, the `PRINT` operator converts the data to the WebRowSet format and sends the results to the client.

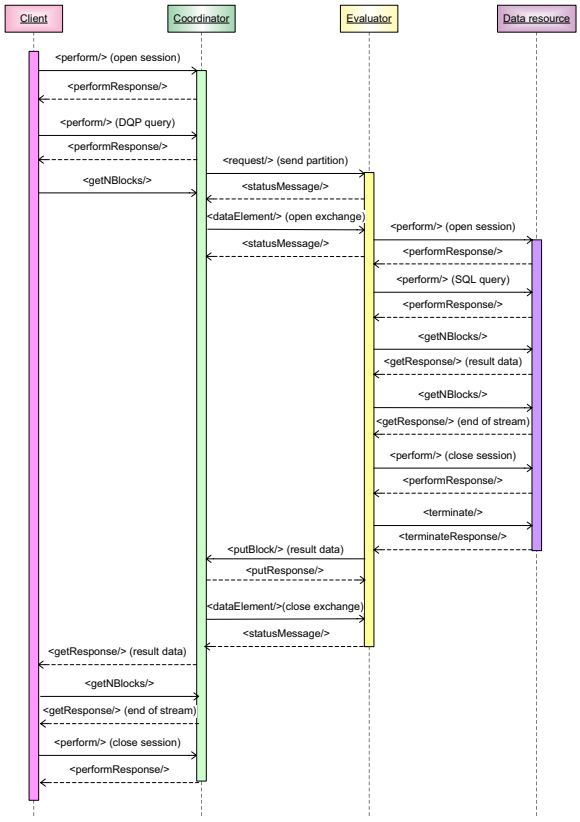

Figure 3. SOAP messages exchanged in query 3

The message exchanges are illustrated as UML sequence diagrams in Figures 3and 4. Figure 3 shows SOAP messages exchanged, and Figure 4 gives a more abstract "reverse-engineered" view, in terms of WSDL-level operations .

The UML sequence diagrams shown in these figures describe the interactions among client, coordinator, evaluator and data resource. The initial exchanges between client and coordinator establish a session and send the OGSA-DQP query, using OGSA-DAI operations. The coordinator compiles the query and sends a partition to the evaluator (in general, the coordinator would send multiple partitions to multiple evaluators). The coordinator will "open" the `EXCHANGE` operation in its evaluator to request a result back. The evaluator executes the `TABLE_SCAN` opera-

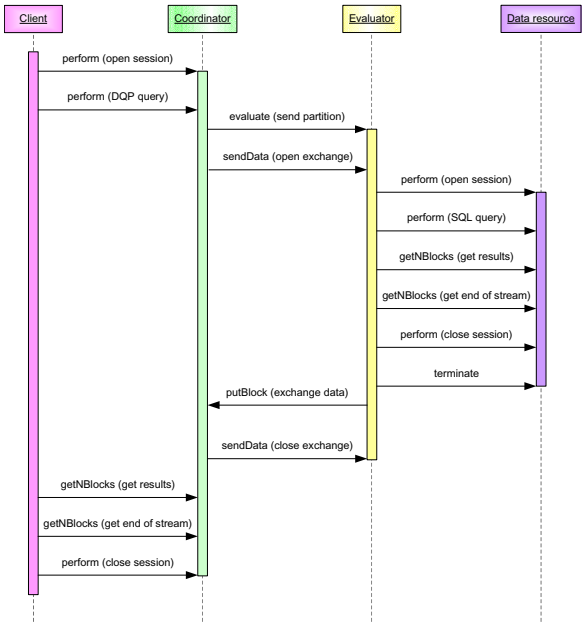

Figure 4. WSDL-level operations in query 3

tor which uses OGSA-DAI operations ([11]) to get the data from the data resource.

The `EXCHANGE` operators return the result data to the coordinator using the OGSA-DAI `putBlock` operation. The coordinator "closes" the `EXCHANGE` operation on the evaluator. Meanwhile, the client will have requested result using the OGSA-DAI `getNBlocks` operation. The client will receive the result in `WebRowSet` format from the `PRINT` operation.

There are 15 SOAP request/response interactions between components for this minimal OGSA-DQP query.

In the end the insight needed to fix our problem comes from the query plan itself. We modified the OGSA-DQP coordinator and changed the date type mapping (see [13]). The OGSA-DQP query submitted to Oracle returns the correct value after the corrections. The OGSA-DQP query 2 (for example) took 4 seconds (time for running a similar query against SDSS `MyBestDR5` without using OGSA-DQP was less than 1 second.)

5. Conclusions

This paper presented an insight into the execution of OGSA-DQP queries through a few OGSA-DQP queries. We looked at the OGSA-DQP query plan for simple queries, and the different SOAP interactions between OGSA-DQP components. Finally, we successfully run OGSA-DQP queries against the distributed SDSS database after modification to the OGSA-DQP coordinator source code. We will discuss running more complicated queries through OGSA-DQP in a follow-up paper [12].

References

[1] M. A. et al. An experience report on designing and building OGSA-DQP: A service based distributed query processor for the Grid.

[2] M. A. et al. OGSA-DQP: A service for distributed querying on the Grid. In E. B. et al., editor, *EDBT*, volume 2992 of *Lecture Notes in Computer Science*, pages 858–861. Springer, 2004.

[3] M. A. et al. Using OGSA-DQP to support scientific applications for the Grid. In P. Herrero, M. S. Prez, and V. Robles, editors, *SAG*, volume 3458 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 2004.

[4] M. A. et al. Experience on performance evaluation with OGSA-DQP. In *Fourth UK e-Science All Hands Meeting*, 2005.

[5] A. Mukherjee and P. Watson. Adding dynamism to OGSA-DQP: Incorporating the DynaSOAr framework in distributed query processing. Technical Report 979, Newcastle University, School of Computing Science, Aug 2006.

[6] The OGSA-DAI project home page. http://www.ogsdai.org.

[7] The OGSA-DQP project. http://www.ogsadai.org/about/ogsa-dqp.

[8] H. Xiang, M. Baker, and R. Nichol. Experiences mirroring and distributing the Sloan Digital Sky Survey. In *Fifth International Conference on Grid and Cooperative Computing Workshops (GCC 2006), Changsha, China*, pages 518–521. IEEE Computer Society, October 2006.

[9] H. X. Xiang. Experiences acquiring and distributing a large scientific database. In *Second International Conference on Future Generation Communication and Networking Symposia*, volume 2, pages 14–19, Washington, DC, USA, December 2008. IEEE Computer Society.

[10] H. X. Xiang. A grid-based distributed database solution for large astronomy datasets. In *International Conference on Computer Science and Software Engineering*, volume 3, pages 66–69, Washington, DC, USA, December 2008. IEEE Computer Society.

[11] H. X. Xiang. *A Grid-based Distributed Database Solution for Large Astronomy Datasets*. PhD thesis, Portsmouth, UK, February 2008.

[12] H. X. Xiang. Experiences running ogsa-dqp queries against a heterogeneous distributed scientific database. In *The Fifteenth International Conference on Parallel and Distributed Systems*, Shenzhen, China, December 2009. IEEE Computer Society.

[13] H. X. Xiang. Supporting complex scientific database schemas in a grid middleware. In *International Conference on Advanced Information Networking and Applications*, volume 0, pages 937–944, Bradford, UK, May 2009. IEEE Computer Society.

[14] D. G. York et al. The Sloan Digital Sky Survey: Technical summary. *Astronomical Journal*, 120:1579–1587, 2000. http://www.sdss.org.